Lycée TOUCHARD WASHINGTON

Section des Techniciens Supérieurs
Cybersécurité Informatique Electronique
Option Informatique et Réseaux

Projet Digicode

~ TP Ctrl 2 ~
Codage Qt
Interfaces utilisateurs - Signaux & Slots

Version 2.1 – Octobre 2024

Conditions de réalisation

Travail individuel

Durée : 3h

Travail à déposer dans un dépôt privé **Projet_Digicode** de votre GitHub. Vous inviterez votre enseignant en tant que collaborateur.

Après chaque étape, transférez votre réalisation sur votre GitHub en indiquant le nom de l'étape dans le commentaire de votre commit.

Vous rendrez également dans un document README.md une présentation du dépôt .

Ressources utilisées :

Matériel	**Logiciel**
Un ordinateur sous linux	Qt Creator Git

1. PRÉSENTATION DU PROJET

Le digicode est une serrure électronique qui se déverrouille en saisissant un code secret sur un clavier numérique. Il est notamment employé sur les portes d'immeubles pour en contrôler l'accès. L'ouverture de la porte reste manuelle, en la poussant, elle se referme d'elle-même. Pour permettre la sortie des résidents, un bouton poussoir dans le hall de l'immeuble permet de déverrouiller la serrure.

Le gestionnaire de l'immeuble peut modifier le code secret. La procédure consiste à entrer le code puis à faire un appui long sur la touche OK. Le nouveau code secret doit être saisi deux fois pour être pris en compte.

Le projet consiste à réaliser une application qui simule le fonctionnement du système. (**la modification du code secret n'est pas demandée**)

2. CRÉATION DU PROJET

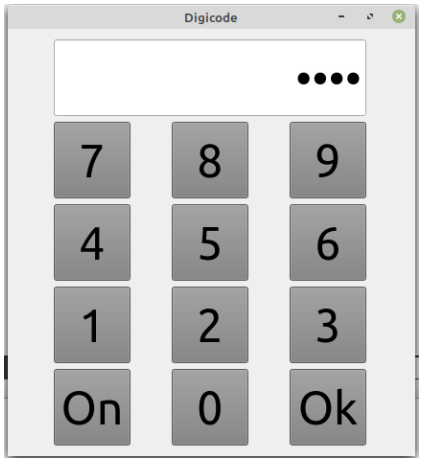
Réalisez dans votre GitHub un dépôt privé nommé **Projet_Digicode**. Il servira de base pour votre projet sous Qt et votre documentation.

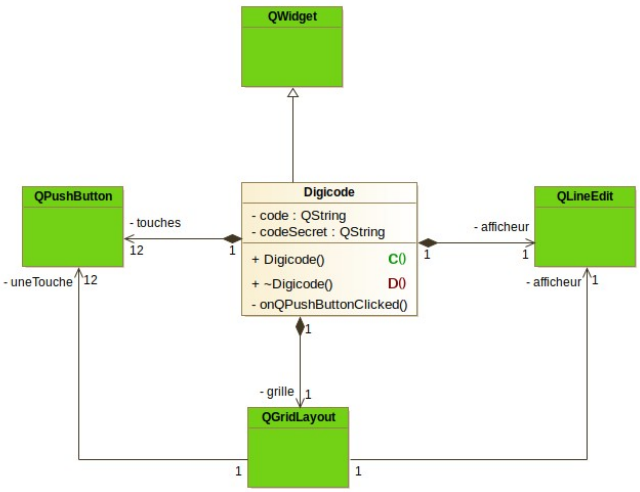
Sous QtCreator, réalisez un projet nommé **digicode** de type ***Qt Widgets Application***. Il sera suivi par le gestionnaire de version **Git**.

3. ÉTAPE N°1 : CRÉATION DE L'INTERFACE UTILISATEUR

Nous allons nous intéresser au clavier du digicode et à la prise en compte des touches du clavier.

Attention toute l'interface sera construit dynamiquement comme le montre les figures suivantes.



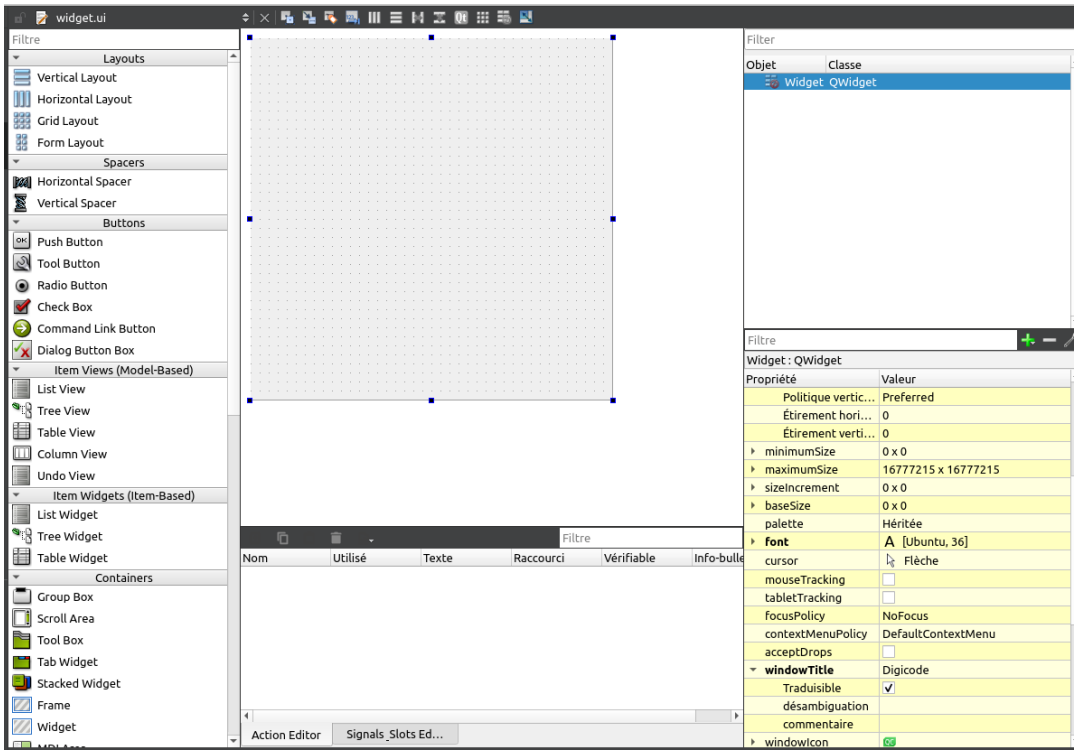


Les touches du clavier se trouveront dans un container de type QGridLayout qui sera créé **dynamiquement** dans le constructeur de digicode comme les autres compositions.

En ce qui concerne les touches du clavier, elles sont réalisées par un tableau de pointeurs sur QPushButton à deux dimensions, 4 lignes 3 colonnes.

Les associations entre ***QGridLayout*** et ***QPushButton*** ou ***QLineEdit*** seront réalisées automatiquement par la méthode ***addWidget()*** de ***QGridLayout***.

Comme le montre la copie d'écran suivante, seules les propriétés **font** et



windowTitle seront définies dans Qt designer

Voici le début du constructeur Digicode

```
ui→setupUi(this);
int colonne=0, ligne=0;
grille = new QgridLayout(this);
afficheur = new QlineEdit(this);
afficheur→setReadOnly(true);
afficheur→setAlignment(Qt::AlignRight);
afficheur→setEchoMode(QLineEdit::Password);
afficheur→setMinimumHeight(80);
grille->addWidget(afficheur,ligne,colonne,1,3);
// Création du clavier
QString TableDesSymboles[4][3] = {{"7","8","9"},{"4","5","6"},
                                  {"1","2","3"},{"On","0","Ok"}};
```

Chaque bouton ne prend qu'une position en ligne et colonne.

On utilisera donc la méthode ***addWidget*** ne comprenant que 3 arguments. En revanche, le placement de l'afficheur (***QLineEdit***) occupe 1 ligne et 3 colonnes. D'où l'utilisation de la méthode ***addWidget*** avec 5 arguments pour la fusion de 3 colonnes. (objet, x,y,rowspan,colspan).

La fin du constructeur est la suivante :

```
this->setLayout(grille);
```

Compléter le constructeur afin de créer les 12 boutons , pour chaque touche :

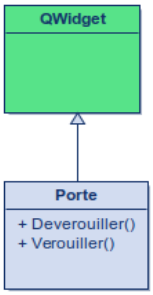
- Mettre le texte correspondant sur chaque bouton : méthode ***setText()***
- La taille des boutons sera définie 80x80 px avec les méthodes setMaximumWidth(80) et setMinimumHeight(80)
- Définir la couleur des touches en gris.
- Connecter l'évènement ***clicked*** au slot ***onQPushButtonClicked()***

Codez la méthode onQPushButtonClicked()

- si la touche cliquée est numérique le texte de la touche est concaténé dans l'attribut code
- si la touche cliquée est la touche OK, le code saisi est comparé au code secret. S'il y a égalité une **QMessageBox** affiche **la porte est déverrouillée.** Dans le cas contraire la ***QMessageBox*** affiche **Code Faux**
- si la touche On est cliquée l'attribut code est vidé.

4. ÉTAPE N°2 : AJOUT DE LA PORTE

Pour simuler le fonctionnement de la porte, créer une classe d'interface graphique nommé **Porte** en respectant le diagramme de classes ci-contre.



Placer sur l'interface ui un QLabel pour afficher l'état de la porte qui peut être soit :

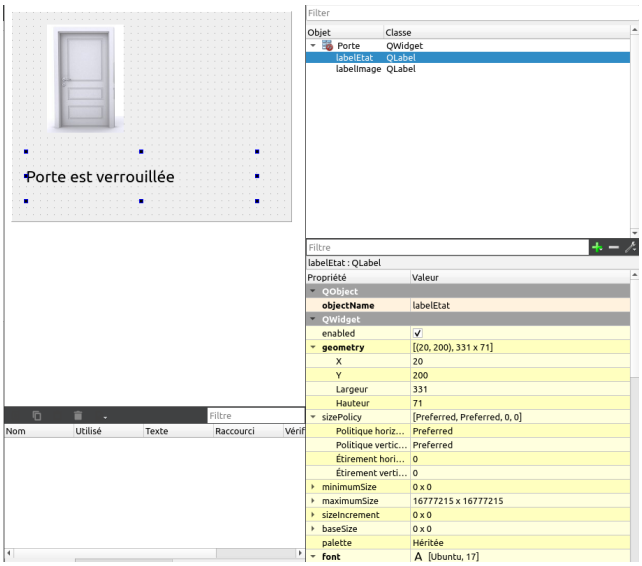
- La porte est déverrouillée
- *La porte est verrouillée*

Un deuxième QLabel contiendra une des images de la porte.

Créer les deux méthodes publiques nommées :

```
void Deverrouiller();
void Verrouiller();
```

Ces deux méthodes changent le texte affiché par le QLabel nommé **labelEtat,** et l'image affiché dans **labelImage** avec la méthode **setPixmap()** de Qlabel.



Voir en annexe la procédure pour ajouter des ressources dans votre projet Qt.

Le constructeur de la classe **Porte** appel la métode Verouiller()

Déclarer une instance **laPorte** comme attribut privé de la classe **Digicode**. La fenêtre qui affiche l'état de la porte apparaît lorsque le code saisi est correct.

5. ÉTAPE N°3 : AJOUT DU TEMPORISATEUR D'ALIMENTATION GÂCHE

La gâche qui autorise l'ouverture de la porte,ne doit être actionnée que pendant un laps de temps limité à 3 secondes.

Ajouter une temporisation tempoGache de type **QTimer** à la classe Digicode.

C'est un temporisateur à un seul coup qui se déclenche une seule fois. Pour ce faire modifier la propriété singleShot pour la mettre à vrai.

Lorsque le temporisateur est écoulé la porte est de nouveau verrouillée lors de sa fermeture.

Déclarer et coder le slot : `void onTimerTempoGache_timeout();`

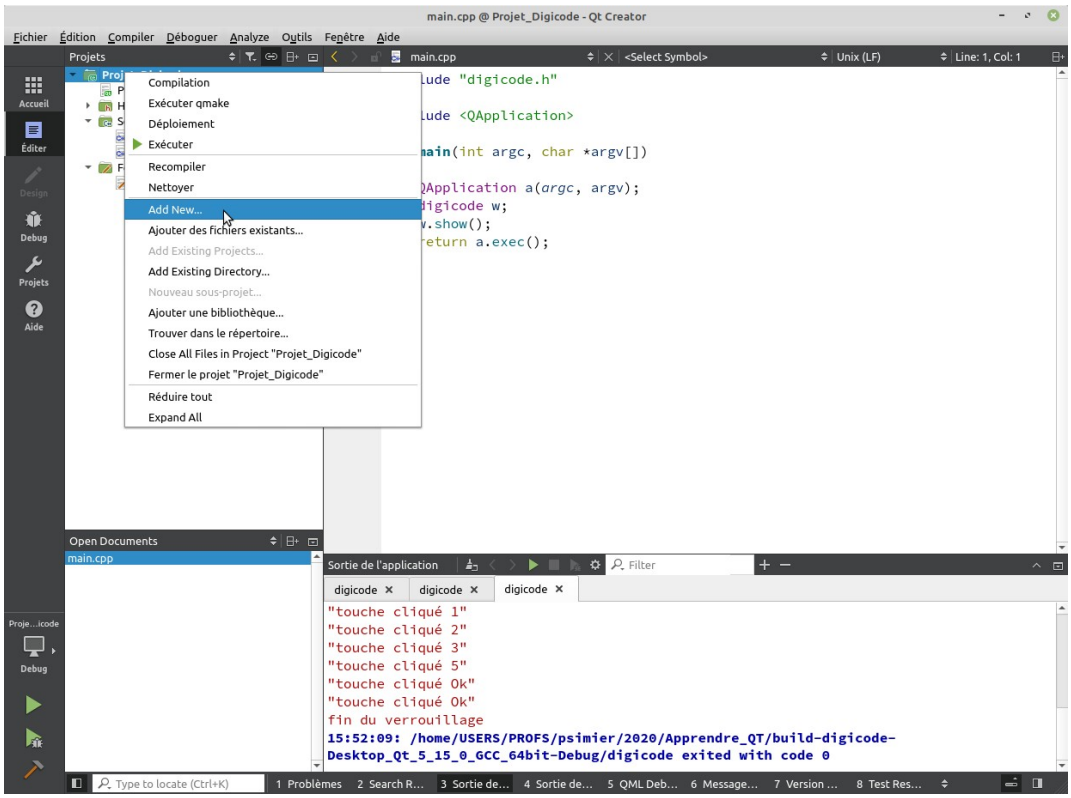
Le slot appelle la méthode Verrouiller() de l'objet laPorte.

6. ANNEXES - LES RESSOURCES

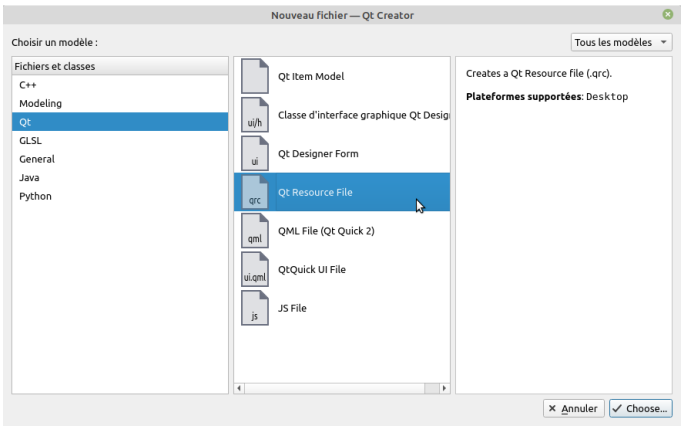
Il est souvent utile de mettre des images et des icônes dans une application. Leur utilisation est très simple dans Qt grâce aux ressources. Le principe des ressources est d'encapsuler les données des images, icônes dans les données du programme, et de les rendre disponibles à toutes les classes grâce à une syntaxe particulière.

Pour créer un fichier de ressource, suivez la procédure suivante :

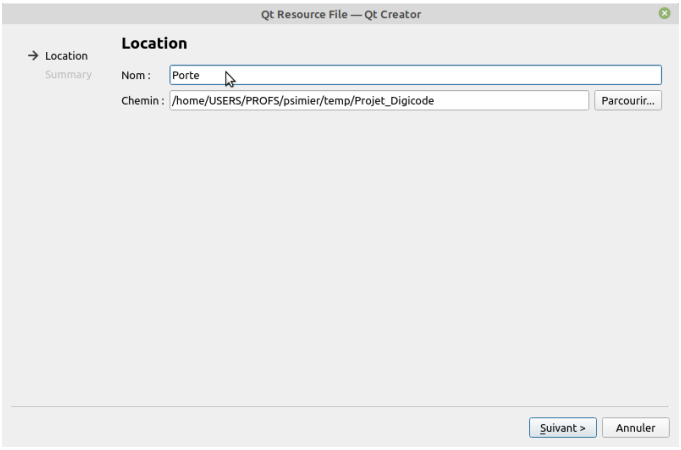
Faire un clique droit sur le nom du projet, puis choisir **Add New...**



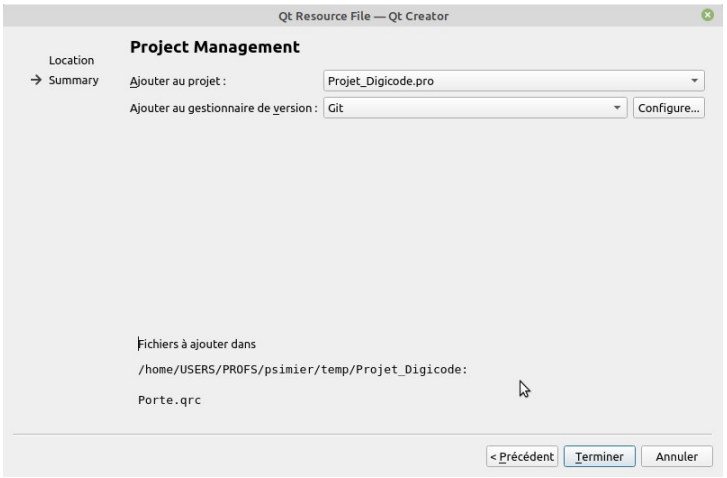
Dans la fenêtre Nouveau fichier : Choisir choisir **Qt** puis **Qt Ressource File**



Cliquer sur le bouton **Choose...** Donner un nom à votre fichier et valider



Dans **Project Management** Ajouter au gestionnaire de version **Git**

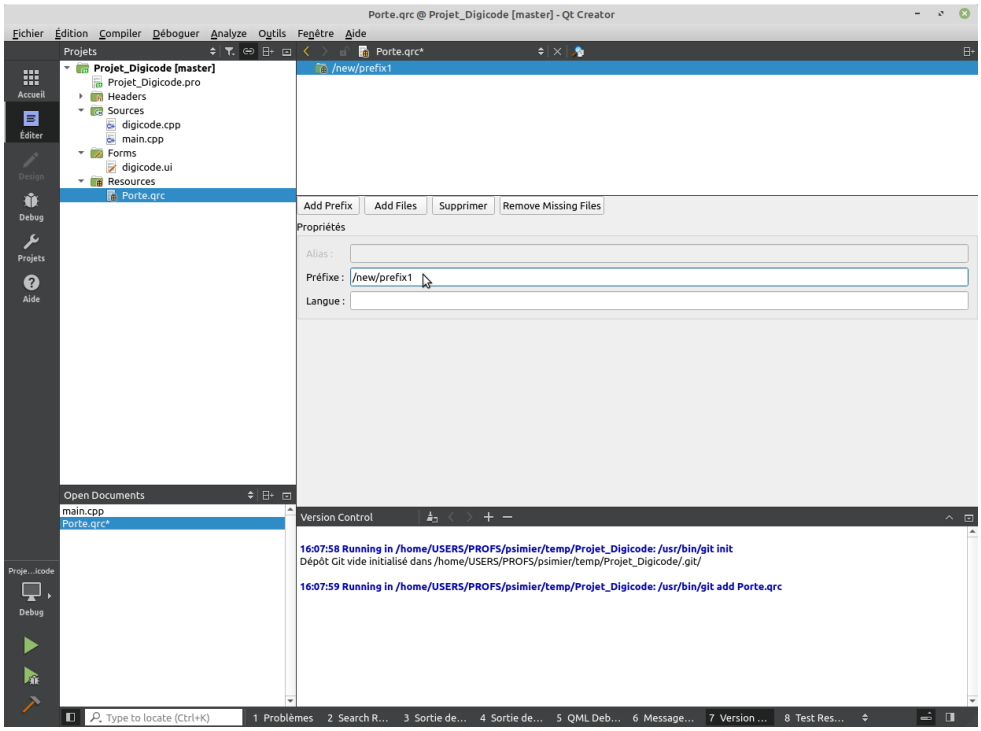


Les ressources peuvent être organisées en groupes grâce à l'ajout d'un préfixe. Au sein d'un préfixe vous pouvez ajouter autant de fichiers que nécessaire.

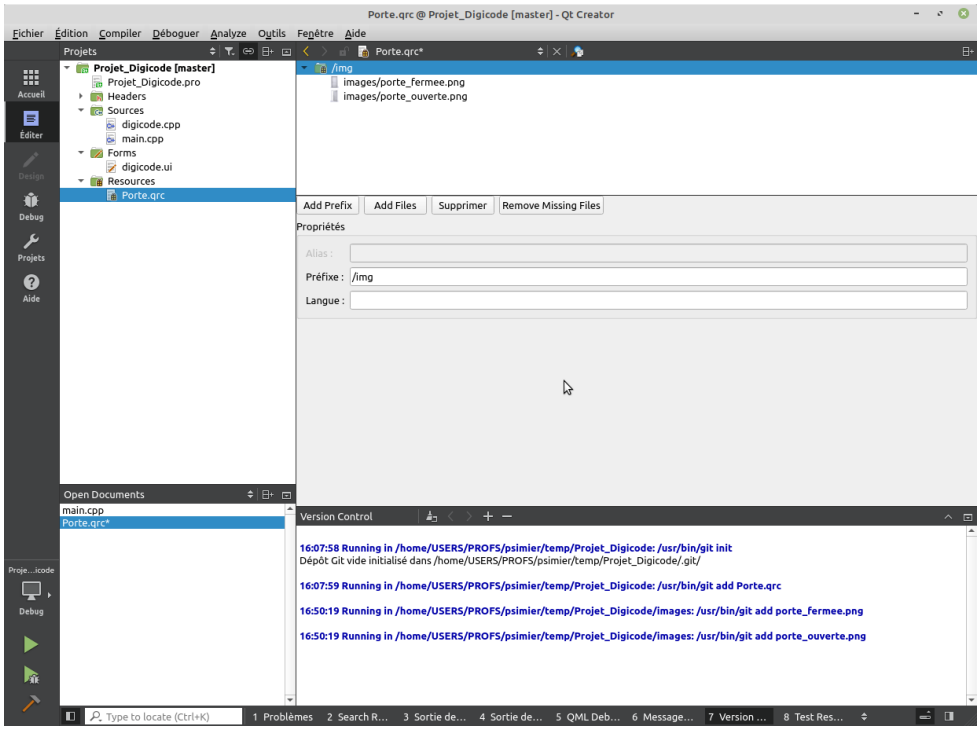
Un alias peut être donné à la ressource car cela facilite l'intégration dans le code.

6.1. *Créer un fichier de ressources*

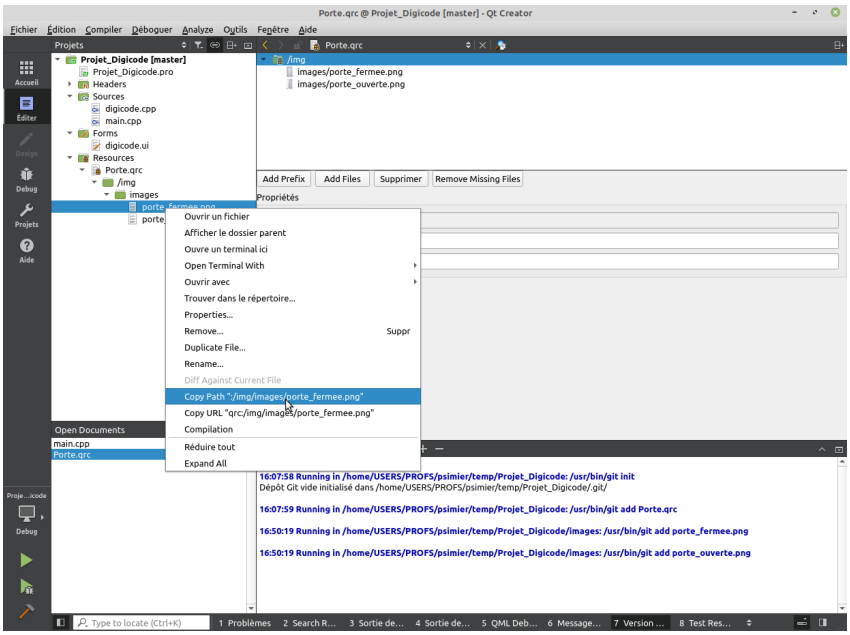
Cliquer sur l'onglet **Add Prefix,** remplacer **/new/prefix1** par **/img**



Cliquer sur l'onglet **Add Files,** sélectionner les fichiers images .png et .jpg à ajouter aux ressources.



Cliquer sur compiler, les images sélectionnées sont maintenant dans l'arborescence du projet comme le montre la capture d'écran suivant.



6.2. *Utiliser une ressource depuis le code*

Pour utiliser une ressource depuis le code source du programme, employer une simple chaîne de caractères, comme si vous indiquiez le chemin d'accès à un fichier.

Dans arborescence des ressources, faire un clique droit sur une des images pour copier le chemin d'accès

vous pouvez passer un chemin de ressource au lieu d'un nom de fichier au constructeur de QIcon, QImage ou QPixmap : comme par exemple

```
ui->labelImage->setPixmap(QPixmap(":/img/images/porte_fermee.png"));
```

Après leur création, les ressources apparaissent dans le fichier .pro

```
SOURCES += \
    main.cpp \
    digicode.cpp
HEADERS += \
    digicode.h
FORMS += \
    digicode.ui
# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
RESOURCES += \
    Porte.qrc
```

Le fichier .qrc est une fichier xml

```
<RCC>
  <qresource prefix="/img">
    <file>images/porte_fermee.png</file>
    <file>images/porte_ouverte.png</file>
  </qresource>
</RCC>
```